

NumPy

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia https://twitter.com/tutorialspoint

About the Tutorial

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

This tutorial explains the basics of NumPy such as its architecture and environment. It also discusses the various array functions, types of indexing, etc. An introduction to Matplotlib is also provided. All this is explained with the help of examples for better understanding.

Audience

This tutorial has been prepared for those who want to learn about the basics and various functions of NumPy. It is specifically useful for algorithm developers. After completing this tutorial, you will find yourself at a moderate level of expertise from where you can take yourself to higher levels of expertise.

Prerequisites

You should have a basic understanding of computer programming terminologies. A basic understanding of Python and any of the programming languages is a plus.

Disclaimer & Copyright

Table of Contents

1. NUMPY – INTRODUCTION

NumPy is a Python package. It stands for 'Numerical Python'. It is a library consisting of multidimensional array objects and a collection of routines for processing of array.

Numeric, the ancestor of NumPy, was developed by Jim Hugunin. Another package Numarray was also developed, having some additional functionalities. In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package. There are many contributors to this open source project.

Operations using NumPy

Using NumPy, a developer can perform the following operations:

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

NumPy – A Replacement for MatLab

NumPy is often used along with packages like **SciPy** (Scientific Python) and **Mat−plotlib** (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing. However, Python alternative to MatLab is now seen as a more modern and complete programming language.

It is open source, which is an added advantage of NumPy.

2. NUMPY – ENVIRONMENT

Standard Python distribution doesn't come bundled with NumPy module. A lightweight alternative is to install NumPy using popular Python package installer, **pip**.

```
pip install numpy
```

The best way to enable NumPy is to use an installable binary package specific to your operating system. These binaries contain full SciPy stack (inclusive of NumPy, SciPy, matplotlib, IPython, SymPy and nose packages along with core Python).

Windows

Anaconda (from https://www.continuum.io) is a free Python distribution for SciPy stack. It is also available for Linux and Mac.

Canopy (https://www.enthought.com/products/canopy/) is available as free as well as commercial distribution with full SciPy stack for Windows, Linux and Mac.

Python (x,y): It is a free Python distribution with SciPy stack and Spyder IDE for Windows OS. (Downloadable from http://python-xy.github.io/)

Linux

Package managers of respective Linux distributions are used to install one or more packages in SciPy stack.

For Ubuntu

```
sudo apt-get install python-numpy python-scipy python-matplotlibipythonipython-
notebook python-pandas python-sympy python-nose
```

For Fedora

```
sudo yum install numpyscipy python-matplotlibipython python-pandas sympy python-
nose atlas-devel
```

Building from Source

Core Python (2.6.x, 2.7.x and 3.2.x onwards) must be installed with distutils and zlib module should be enabled.

GNU gcc (4.2 and above) C compiler must be available.

To install NumPy, run the following command.

```
Python setup.py install
```

To test whether NumPy module is properly installed, try to import it from Python prompt.

```
import numpy
```

If it is not installed, the following error message will be displayed.

```
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    import numpy
ImportError: No module named 'numpy'
```

Alternatively, NumPy package is imported using the following syntax:

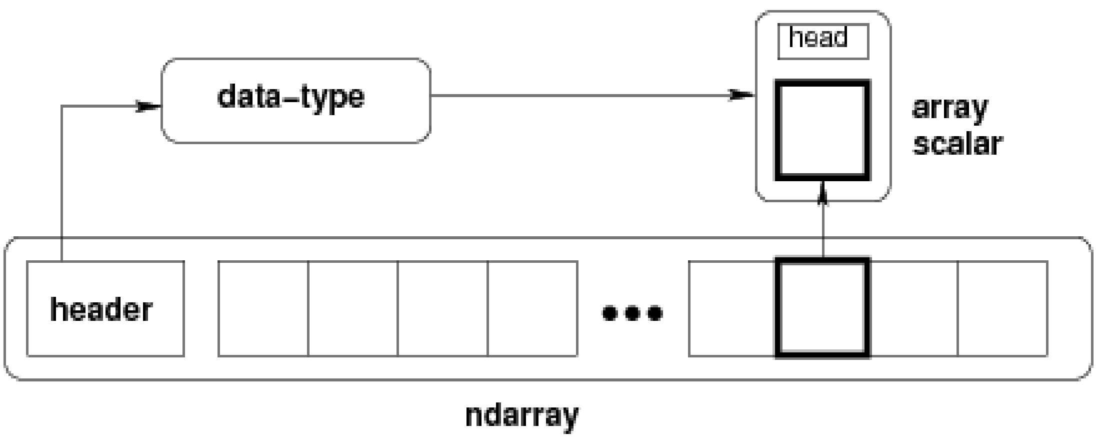
```
import numpy as np
```

3. NUMPY – NDARRAY OBJECT

The most important object defined in NumPy is an N-dimensional array type called **ndarray**. It describes the collection of items of the same type. Items in the collection can be accessed using a zero-based index.

Every item in an ndarray takes the same size of block in the memory. Each element in ndarray is an object of data-type object (called **dtype**).

Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types. The following diagram shows a relationship between ndarray, data type object (dtype) and array scalar type:



An instance of ndarray class can be constructed by different array creation routines described later in the tutorial. The basic ndarray is created using an array function in NumPy as follows:

```
numpy.array
```

It creates an ndarray from any object exposing array interface, or from any method that returns an array.

```
numpy.array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)
```

The above constructor takes the following parameters:

object	Any object exposing the array interface method returns an array, or any (nested) sequence
dtype	Desired data type of array, optional
copy	Optional. By default (true), the object is copied
order	C (row major) or F (column major) or A (any) (default)
subok	By default, returned array forced to be a base class array. If true, sub-classes passed through
ndimin	Specifies minimum dimensions of resultant array

Take a look at the following examples to understand better.

Example 1

```
import numpy as np
a=np.array([1,2,3])
print a
```

The output is as follows:

```
[1, 2, 3]
```

Example 2

```
# more than one dimensions
import numpy as np
```

```
a = np.array([[1, 2], [3, 4]])
print a
```

The output is as follows:

```
[[1, 2]
 [3, 4]]
```

Example 3

```
# minimum dimensions
import numpy as np
a=np.array([1, 2, 3,4,5], ndmin=2)
print a
```

The output is as follows:

```
[[1, 2, 3, 4, 5]]
```

Example 4

```
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype=complex)
print a
```

The output is as follows:

```
[ 1.+0.j,  2.+0.j,  3.+0.j]
```

The **ndarray** object consists of contiguous one-dimensional segment of computer memory, combined with an indexing scheme that maps each item to a location in the memory block. The memory block holds the elements in a row-major order (C style) or a column-major order (FORTRAN or MatLab style).

4. NUMPY – DATA TYPES

NumPy supports a much greater variety of numerical types than Python does. The following table shows different scalar data types defined in NumPy.

Data Types	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

NumPy numerical types are instances of dtype (data-type) objects, each having unique characteristics. The dtypes are available as np.bool_, np.float32, etc.

Data Type Objects (dtype)

A data type object describes interpretation of fixed block of memory corresponding to an array, depending on the following aspects:

- Type of data (integer, float or Python object)
- Size of data
- Byte order (little-endian or big-endian)
- In case of structured type, the names of fields, data type of each field and part of the memory block taken by each field
- If data type is a subarray, its shape and data type

The byte order is decided by prefixing '<' or '>' to data type. **'<'** means that encoding is little-endian (least significant is stored in smallest address). **'>'** means that encoding is big-endian (most significant byte is stored in smallest address).

A dtype object is constructed using the following syntax:

```
numpy.dtype(object, align, copy)
```

The parameters are:

- **Object**: To be converted to data type object
- **Align**: If true, adds padding to the field to make it similar to C-struct
- **Copy**: Makes a new copy of dtype object. If false, the result is reference to built-in data type object

Example 1

```
# using array-scalar type
import numpy as np
dt=np.dtype(np.int32)
print dt
```

The output is as follows:

```
int32
```

Example 2

```
#int8, int16, int32, int64 can be replaced by equivalent string 'i1', 'i2','i4', etc.
import numpy as np
dt = np.dtype('i4')
print dt
```

The output is as follows:

```
int32
```

Example 3

```
# using endian notation
import numpy as np
dt = np.dtype('>i4')
print dt
```

The output is as follows:

```
>i4
```

The following examples show the use of structured data type. Here, the field name and the corresponding scalar data type is to be declared.

Example 4

```
# first create structured data type
import numpy as np
dt = np.dtype([('age',np.int8)])
print dt
```

The output is as follows:

```
[('age', 'i1')]
```

Example 5

```
# now apply it to ndarray object
```

```
import numpy as np
dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype=dt)
print a
```

The output is as follows:

```
[(10,) (20,) (30,)]
```

Example 6

```
# file name can be used to access content of age column
import numpy as np
dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype=dt)
print a['age']
```

The output is as follows:

```
[10 20 30]
```

Example 7

The following examples define a structured data type called **student** with a string field 'name', an **integer field** 'age' and a **float field** 'marks'. This dtype is applied to ndarray object.

```
import numpy as np
student=np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
print student
```

The output is as follows:

```
[('name', 'S20'), ('age', 'i1'), ('marks', '<f4')])
```

Example 8

```
import numpy as np
student=np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype=student)
```

```
print a
```

The output is as follows:

```
[('abc', 21, 50.0), ('xyz', 18, 75.0)]
```

Each built-in data type has a character code that uniquely identifies it.

- **'b'**: boolean
- **'i'**: (signed) integer
- **'u'**: unsigned integer
- **'f'**: floating-point
- **'c'**: complex-floating point
- **'m'**: timedelta
- **'M'**: datetime
- **'O'**: (Python) objects
- **'S', 'a'**: (byte-)string
- **'U'**: Unicode
- **'V'**: raw data (void)

5. NUMPY – ARRAY ATTRIBUTES

In this chapter, we will discuss the various array attributes of NumPy.

ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

Example 1

```
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
print a.shape
```

The output is as follows:

```
(2, 3)
```

Example 2

```
# this resizes the ndarray
import numpy as np
a=np.array([[1,2,3],[4,5,6]])
a.shape=(3,2)
print a
```

The output is as follows:

```
[[1  2]
 [3  4]
 [5  6]]
```

Example 3

NumPy also provides a reshape function to resize an array.

```
import numpy as np
```

```
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print b
```

The output is as follows:

```
[[1  2]
 [3  4]
 [5  6]]
```

ndarray.ndim

This array attribute returns the number of array dimensions.

Example 4

```
# an array of evenly spaced numbers
import numpy as np
a = np.arange(24)
print a
```

The output is as follows:

```
[0 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 17 18 19 20 21 22 23]
```

Example 5

```
# this is one dimensional array
import numpy as np
a = np.arange(24)
a.ndim

# now reshape it
b = a.reshape(2,4,3)
print b
# b is having three dimensions
```

The output is as follows:

```
[[[ 0,  1,  2]
  [ 3,  4,  5]
  [ 6,  7,  8]
  [ 9, 10, 11]]

  [[12, 13, 14]
   [15, 16, 17]
   [18, 19, 20]
   [21, 22, 23]]]
```

numpy.itemsize

This array attribute returns the length of each element of array in bytes.

Example 6

```
# dtype of array is int8 (1 byte)
import numpy as np
x = np.array([1,2,3,4,5], dtype=np.int8)
print x.itemsize
```

The output is as follows:

```
1
```

Example 7

```
# dtype of array is now float32 (4 bytes)
import numpy as np
x = np.array([1,2,3,4,5], dtype=np.float32)
print x.itemsize
```

The output is as follows:

```
4
```

numpy.flags

The ndarray object has the following attributes. Its current values are returned by this function.

C_CONTIGUOUS (C)	The data is in a single, C-style contiguous segment
F_CONTIGUOUS (F)	The data is in a single, Fortran-style contiguous segment
OWNDATA (O)	The array owns the memory it uses or borrows it from another object
WRITEABLE (W)	The data area can be written to. Setting this to False locks the data, making it read-only
ALIGNED (A)	The data and all elements are aligned appropriately for the hardware
UPDATEIFCOPY (U)	This array is a copy of some other array. When this array is deallocated, the base array will be updated with the contents of this array

Example 8

The following example shows the current values of flags.

```
import numpy as np
x = np.array([1,2,3,4,5])
print x.flags
```

The output is as follows:

```
C_CONTIGUOUS : True
F_CONTIGUOUS : True
OWNDATA : True
WRITEABLE : True
ALIGNED : True
UPDATEIFCOPY : False
```

6. NUMPY – ARRAY CREATION ROUTINES

A new **ndarray** object can be constructed by any of the following array creation routines or using a low-level ndarray constructor.

numpy.empty

It creates an uninitialized array of specified shape and dtype. It uses the following constructor:

```
numpy.empty(shape, dtype=float, order='C')
```

The constructor takes the following parameters.

Shape	Shape of an empty array in int or tuple of int
Dtype	Desired output data type. Optional
Order	'C' for C-style row-major array, 'F' for FORTRAN style column-major array

Example 1

The following code shows an example of an empty array.

```
import numpy as np
x = np.empty([3,2], dtype=int)
print x
```

The output is as follows:

```
[[22649312    1701344351]
 [1818321759  1885959276]
 [16779776    156368896]]
```

Note: The elements in an array show random values as they are not initialized.

numpy.zeros

Returns a new array of specified size, filled with zeros.

```
numpy.zeros(shape, dtype=float, order='C')
```

The constructor takes the following parameters.

Shape	Shape of an empty array in int or sequence of int
Dtype	Desired output data type. Optional
Order	'C' for C-style row-major array, 'F' for FORTRAN style column-major array

Example 2

```
# array of five zeros. Default dtype is float
import numpy as np
x = np.zeros(5)
print x
```

The output is as follows:

```
[ 0.  0.  0.  0.  0.]
```

Example 3

```
import numpy as np
x = np.zeros((5,), dtype=np.int)
print x
```

Now, the output would be as follows:

```
[0  0  0  0  0]
```

Example 4

```
# custom type
import numpy as np
x = np.zeros((2,2), dtype=[('x', 'i4'), ('y', 'i4')])
print x
```

It should produce the following output:

```
[[(0, 0) (0, 0)]
 [(0, 0) (0, 0)]]
```

numpy.ones

Returns a new array of specified size and type, filled with ones.

```
numpy.ones(shape, dtype=None, order='C')
```

The constructor takes the following parameters.

Shape	Shape of an empty array in int or tuple of int
Dtype	Desired output data type. Optional
Order	'C' for C-style row-major array, 'F' for FORTRAN style column-major array

Example 5

```
# array of five ones. Default dtype is float
import numpy as np
x = np.ones(5)
print x
```

The output is as follows:

```
[ 1.  1.  1.  1.  1.]
```

Example 6

```
import numpy as np
x = np.ones([2,2], dtype=int)
print x
```

Now, the output would be as follows:

```
[[1  1]
 [1  1]]
```

7. NUMPY – ARRAY FROM EXISTING DATA

In this chapter, we will discuss how to create an array from existing data.

numpy.asarray

This function is similar to numpy.array except for the fact that it has fewer parameters. This routine is useful for converting Python sequence into ndarray.

```
numpy.asarray(a, dtype=None, order=None)
```

The constructor takes the following parameters.

a	Input data in any form such as list, list of tuples, tuples, tuple of tuples or tuple of lists
dtype	By default, the data type of input data is applied to the resultant ndarray
order	C (row major) or F (column major). C is default

The following examples show how you can use the **asarray** function.

Example 1

```
# convert list to ndarray
import numpy as np
x = [1,2,3]
a = np.asarray(x)
print a
```

Its output would be as follows:

```
[1  2  3]
```

Example 2

```
# dtype is set
import numpy as np
x = [1,2,3]
a = np.asarray(x, dtype=float)
print a
```

Now, the output would be as follows:

```
[ 1.  2.  3.]
```

Example 3

```
# ndarray from tuple
import numpy as np
x = (1,2,3)
a = np.asarray(x)
print a
```

Its output would be:

```
[1  2  3]
```

Example 4

```
# ndarray from list of tuples
import numpy as np
x = [(1,2,3),(4,5)]
a = np.asarray(x)
print a
```

Here, the output would be as follows:

```
[(1, 2, 3) (4, 5)]
```

numpy.frombuffer

This function interprets a buffer as one-dimensional array. Any object that exposes the buffer interface is used as parameter to return an **ndarray**.

```
numpy.frombuffer(buffer, dtype=float, count=-1, offset=0)
```

The constructor takes the following parameters.

buffer	Any object that exposes buffer interface
dtype	Data type of returned ndarray. Defaults to float
count	The number of items to read, default -1 means all data
offset	The starting position to read from. Default is 0

Example 5

The following examples demonstrate the use of **frombuffer** function.

```
import numpy as np

s = 'Hello World'

a = np.frombuffer(s, dtype='S1')

print a
```

Here is its output:

```
['H'  'e'  'l'  'l'  'o'  ' '  'W'  'o'  'r'  'l'  'd']
```

numpy.fromiter

This function builds an **ndarray** object from any iterable object. A new one-dimensional array is returned by this function.

```
numpy.fromiter(iterable, dtype, count=-1)
```

Here, the constructor takes the following parameters.

iterable	Any iterable object
dtype	Data type of resultant array

count	The number of items to be read from iterator. Default is -1 which means all data to be read

The following examples show how to use the built-in **range()** function to return a list object. An iterator of this list is used to form an **ndarray** object.

Example 6

```
# create list object using range function
import numpy as np
list = range(5)
print list
```

Its output is as follows:

```
[0,  1,  2,  3,  4]
```

Example 7

```
# obtain iterator object from list
import numpy as np
list = range(5)
it = iter(list)

# use iterator to create ndarray
x = np.fromiter(it, dtype=float)
print x
```

Now, the output would be as follows:

```
[0.   1.   2.   3.   4.]
```

8. NUMPY – ARRAY FROM NUMERICAL RANGES

In this chapter, we will see how to create an array from numerical ranges.

numpy.arange

This function returns an **ndarray** object containing evenly spaced values within a given range. The format of the function is as follows:

```
numpy.arange(start, stop, step, dtype)
```

The constructor takes the following parameters.

start	The start of an interval. If omitted, defaults to 0
stop	The end of an interval (not including this number)
step	Spacing between values, default is 1
dtype	Data type of resulting ndarray. If not given, data type of input is used

The following examples show how you can use this function.

Example 1

```
import numpy as np
x = np.arange(5)
print x
```

Its output would be as follows:

```
[0  1  2  3  4]
```

Example 2

```
import numpy as np
# dtype set
x = np.arange(5, dtype=float)
print x
```

Here, the output would be:

```
[0.  1.  2.  3.  4.]
```

Example 3

```
# start and stop parameters set
import numpy as np
x = np.arange(10,20,2)
print x
```

Its output is as follows:

```
[10  12  14  16  18]
```

numpy.linspace

This function is similar to **arange()** function. In this function, instead of step size, the number of evenly spaced values between the interval is specified. The usage of this function is as follows:

```
numpy.linspace(start, stop, num, endpoint, retstep, dtype)
```

The constructor takes the following parameters.

start	The starting value of the sequence
stop	The end value of the sequence, included in the sequence if endpoint set to true

num	The number of evenly spaced samples to be generated. Default is 50
endpoint	True by default, hence the stop value is included in the sequence. If false, it is not included
retstep	If true, returns samples and step between the consecutive numbers
dtype	Data type of output **ndarray**

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**